BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

(12)
B.S.

BBN Report No. 3266

March 1976

THE FOREMAN: Providing the Program Execution Environment for the National Software Works

Richard E. Schantz

Robert E. Millstein

Preliminary
March 31, 1976

See 1473

This work was supported by the Defense Advanced Research Projects Agency of the Department of Defense and by Rome Air Development Center. It was monitored by Rome Air Development Center under Contract Number F30602-76-C-0094 and by the Office of Naval Research under Contract Number N0014-75-C-0773.

The Foreman: Providing the Program Execution Environment for the National Software Works

Richard E. Schantz
Bolt Beranek and Newman Inc.

Robert E. Millstein
Massachusetts Computer Associates Inc.

Preliminary
March 31, 1976

Bolt Beranek and Newman Inc. Report No. 3266
Massachusetts Computer Associates Inc. Document No. CADD-7604-011

This work was supported by the Advanced Research Projects Agency of the Department of Defense and by Rome Air Development Center. It was monitored by Rome Air Development Center under contract number F30602-76-C-0094 and by the Office of Naval Research under contract number N0014-75-C-0773.

Acknowledgement

The authors wish to thank Robert H. Thomas of BBN for help in developing this document.

Table of Contents

The Foreman: Providing the Program Execution Environment for the National Software Works

I. Introduction

1.1 Overview

The National Software Works (NSW) is a facility resident on the ARPANET, principally intended to support the construction of computer programs and to provide software tools (e.g., editors, compilers, debuggers, etc.) which can be used in the construction activity. A prominent factor in the conception of NSW is the expectation that the hardware, software, and human resources needed for the execution of a task may be geographically and administratively dispersed, although connected through the network. As such, the NSW is a distributed, multi-computer system.

The major components of the NSW are the ARPANET, a collection of tool-bearing hosts, one or more front-end (FE) systems through which the users access the NSW, and an access-granting, resource-allocating central component called a Works Manager (WM). A tool-bearing host (TBH) is a computer system which houses software development tools made available to users through the NSW, and which additionally may provide storage for NSW user files. To become a TBH, a host must implement a set of supervisory software modules. Included in this set are the modules which handle the NSW communication needs (see MSG: The Interprocess Communication Facility for the NSW, Bolt Beranek and Newman Inc. Report No. 3237 and Massachusetts Computer Associates Inc. Document No. CADD-7601-2611), the modules which handle NSW file transfer requirements (see File Package: The File Handling Facility for the NSW, Mass. Computer Associates Document No. CADD-7602-2011), and the modules which provide the local host functions for invoking and controlling NSW tools, as well as providing a local host NSW tool execution environment. These latter tool-oriented modules are collectively known as the Foreman component of the NSW, and are the subject of this document.

The Foreman specification is especially intended for those persons responsible for implementing TBH Foremen. In many cases, we are merely presenting approaches to problems which must be faced. While we do require that most (if not all) of the functionality we will specify be made available to the tools, the form in which it is presented to the tool is a purely local decision. The only strict requirement is that each Foreman must implement the various functions which can be invoked externally (see Appendix 1). The exact time and sequence for the Foreman itself invoking functions in the other NSW components (e.g., WM) is discretionary. In most areas, this document also suggests implementation strategies which we feel are worthwhile. However, because of the discretionary nature of the Foreman/tool interface, aspects of this interface mentioned in this document should not be considered as final specification to the tool builder. Each Foreman implementer is responsible for providing the exact details of his tool interface through a host specific tool builder's guide. Potential tool builder's should contact NSW TBH personnel for the tool's implementation host before beginning to integrate their tool(s) into the NSW.

1.2 The Role of the Foreman

The Foreman is the local-to-the-tool component of the NSW system. Each instance of a tool has a Foreman which is responsible for the smooth operation of the tool with the other NSW components. In conjunction with the facilities supported by the Works Manager (WM) component, and to a lesser extent the Front End (FE) component, the Foreman provides the tool instance with its NSW execution environment. Every tool instance runs under the control of a Foreman. Components which do not run under a Foreman are considered part of the implementation of the NSW structure itself.

Note carefully the distinction between the concept of a "tool" and the concept of a "tool instance". A tool refers to a computer program, while a tool instance corresponds to the abstract notion of a process. A tool is a static concept in NSW, while a tool instance is dynamic. The difference is also reflected in the nature of the information maintained by the NSW about each. A tool has a static tool descriptor, maintained by the WM, which describes information pertinent to each use of the tool. Information pertaining to a tool instance is maintained in a dynamic data base, partly within the WM and partly within the Foreman. The dynamic entries referring to a tool instance are maintained for the duration of the tool session only. In this specification we shall often simply refer to "the tool" instead of "the tool instance", in order to avoid the constant repetition of the word instance. However, it should be clear from the context whether we are referring to the static concept of a tool as a program (hardly ever) or the dynamic concept of a tool instance as the execution of that program (almost always).

The Foreman has two well defined parallel interfaces, both of which are described in this specification. One interface is between the Works Manager processes and the Foreman. This interface is organized around the MSG message passing capability, and involves both the WM instructing the Foreman about handling the tool instance, and the Foreman requesting WM services on behalf of its tool. The responses to these commands/requests are obviously also part of the WM/Foreman message interface. The other well defined interface is between the Foreman and its tool instance. The Foreman has a special relationship with the tool it is monitoring. In addition to having the responsibility for creating and subsequently removing the actual instance of the tool, the Foreman maintains an operating-system-like interface to the tool. It is through this interface that the tool can invoke the various functions provided by the NSW environment to augment the host operating system environment. The tool/Foreman interface can take any of a number of forms, the selection made by the implementer of the Foreman for a particular host. Examples of the various types of tool/Foreman linkage include subroutine call (as in MULTICS), operating system call (SVC in the IBM series, JSYS in TENEX), and short messages (ELF). A host-specific tool implementer's guide will detail the exact nature of the tool/Foreman linkage (for each system) as well as the exact nature of the NSW system calls available to tools on that host. In this document, we attempt to specify the functions which the Foreman is expected to implement, help implement, or provide access to. Some implementation details are given. In addition, we mention an optional Foreman function (encapsulation) which can greatly ease the task of bringing into the NSW selected tools which already exist as local host programs.

This document should be viewed as the first of a series of specifications of functions to be performed by the Foreman. The initial tools do not require sophisticated system support, nor has there been adequate time to fully investigate important areas such as error recovery. Because of this, and because of the phased implementation plan for the NSW, this document is vague in some functional areas, and incomplete in others. (It is clearly noted where we intended to have functions incompletely specified. Other instances of this are oversights, and should be immediately noted in any response to this specification.) Future revisions of the Foreman specification will clarify and further define these areas. In addition, as the NSW concept and the NSW system evolve, extensions to the capabilities supported by the Foreman can be expected. As a statement of intent, we feel that it is a sound design strategy to provide tools with mechanisms for doing (almost) all of the things a user at his terminal could do. The initial specification of the Foreman only partially reflects this goal.

It is important to emphasize that the NSW project is not involved in or based on writing new operating systems. Rather, we are building the NSW by allocating subsets of the resources of the participating machines, and using the existing operating system to help manipulate these resources (and at times transform them into resources more appropriate to the NSW environment). This approach is obvious upon examination of the abstract machine under which tool instances are run. The tool environment is a blend of the environment originally provided by the host operating system, augmented in selected areas with facilities implemented by NSW components. Sometimes it will be necessary to mold existing local host facilities into new specifications. This is the case when we utilize an existing local file system capability to support access to files under the NSW system. Other times the NSW facilities will be completely new to a host environment, so that adopting NSW standards directly in the operating system is a possibility. Such may be the case with an MSG communication facility. Accordingly, the structure and flexibility of the existing operating system will greatly influence the structure of a Foreman component for any machine. In specifying the role of the Foreman, we have tried to be as independent as possible of the structure of any operating system and avoid reliance on particular features of any system. We hope an implementation is not only possible but reasonably efficient on a wide variety of operating systems.

1.3 Aspects of the Foreman

There are five aspects of the functioning Foreman which are of concern in this document. They are:

* providing for tool startup/control/termination

* providing the NSW runtime environment for tools written to function in the NSW

* (optionally) providing for encapsulation of programs written exclusively for the local host operating system by defining mappings from existing local operating system functions to NSW system functions

* providing for batch type tools

* providing mechanisms for debugging tools and recovering from errors and malfunctions

Additionally, each tool must be prevented from interfering with other tools and other processes running on the host operating system. Toward this end, we envision a protection domain surrounding the tool, and a temporary workspace for file manipulation during a tool session. In some implementations the host operating system may provide support for these requirements (e.g. separate work directories temporarily assigned to the tool for the duration of the tool session, and subsequently cleared and used by other tools). Where the host operating system does not provide such support, the Foremen themselves must assure tool separation and maintain boundaries between workspaces.

1.4 Short Scenario of Beginning an NSW Session

At this point, we provide a scenario of the beginning of a typical NSW session. It serves to illustrate the roles of the various NSW components and sets a proper context for the discussions to follow. The reader is assumed familiar with the documents referred to above, and in addition the document Works Manager Procedures, reproduced within Mass. Computer Associates document CADD-7603-0411 (also available separately as MCA document CADD-7603-0412).

In this scenario we assume that there exists an NSW process in the (local to the user) Front End machine which is receptive to an attention character on a terminal. We also assume that WM command interpretation is done within this FE process. Our scenario begins from the point where the user has a dedicated FE MSG process assigned to handle his terminal port.

The FE process starts by prompting the user for his login information. After accumulating the pertinent information, the FE process sends the data to a Works Manager process using the generic addressing facility of MSG. The WM receiving the login message will verify the login parameters and note the full name of the FE process servicing this user. (The return address on the message indicates the FE process name.) A specifically addressed message from the WM process to the FE process communicates the success or failure of the user login.

If we assume a successful login, then, when the user wants to run a tool, the FE gathers the name of the tool along with any other pertinent information and sends the request generically to any WM. The request is actually one in which the FE process is asking the WM to establish a new instance of the tool on behalf of the NSW user. The receiving WM verifies whether the user can indeed run such a tool, and, if so, retrieves the tool descriptor from an internal WM data base. Based on information accessible to it, the WM formulates a message containing the tool information and sends it generically to a Foreman process on the host which has been selected (by the WM) to run the tool instance. This information includes the nature of the tool (e.g. encapsulated, uses MSG directly, etc.) and the MSG process name of the FE process for this user. The Foreman selects a workspace for the tool and establishes the tool instance in this workspace. The Foreman then returns (to the WM which called it) the MSG name of the tool which was created. This is done using the specific send MSG capability. The tool name may be different from the Foreman MSG name in the case of non-encapsulated tools using MSG facilities. The WM then replies to the original calling FE process with a specifically addressed message which indicates the MSG addresses of the tool process and the Foreman process.

For the case where the tool and FE communicate via MSG messages, our scenario is complete since both the FE and tool/Foreman have each other's specific name and can send messages directly to each other. For the case where the tool is encapsulated, or where the tool requests to use a network Telnet connection to the FE, the Foreman sends a message to the FE indicating that a direct FE-Tool connection is needed. Using each other's specific names, the FE and tool/Foreman use MSG primitives to establish a direct communication path. As soon as the MSG connection requests match, and the connections are established, data can flow from the user to the tool and vice-versa.

1.5 Conventions Used in this Document

To avoid confusion arising from the ambiguous nature of the names of the various functions implemented within and for the Works Manager, the Foreman and the tool, we adopt a convention within this document to help the reader understand which component implements a given function. We are using a prefix of F$ to indicate an externally callable function implemented within the Foreman (e.g. F$BEGINTOOL), a prefix of W$ to indicate a function within the Works Manager (e.g. W$DELIVER) and all capitals with no $ prefix to indicate Foreman-implemented, tool-callable primitives (e.g. DELETELOCAL). The F$ and W$ prefixes are only used as expository aides, and are not part of the actual function name in either the Foreman or the WM. The actual function name is the string remaining after stripping off the F$ and W$, as appropriate. Interprocess request transmission conventions are currently those specified by Jon Postel (SRI) in his network message of 10 March, 1976. These requests utilize a modified PCPB8 format. However, this format is subject to further changes. The exact specification of the arguments to be sent to a Foreman and returned from a Foreman are compiled in Appendix 1 of this document.

II. Controlling the Execution of a Tool

2.1 Initiating an Instance of a Tool

The MSG facility itself is responsible for the allocation of Foreman processes in response to demand for their use. Thus we can begin our specification assuming the existence of the Foreman as an MSG process, and control lying within the Foreman. After initialization, a Foreman must be receptive to a F$BEGINTOOL message from a Works Manager process. The F$BEGINTOOL message is sent as a generic message addressed to a Foreman of the proper variety (i.e. host type), and can be received via the Receivegeneric capability supported by MSG. The Receivegeneric by the Foreman indicates that it is ready to support a new instance of some tool, on command from a Works Manager. The F$BEGINTOOL message contains a host specific name of the tool to be run. (The WM retrieves the host specific name for this tool from a static tool descriptor it maintains for each tool.) On the basis of this name, the Foreman is expected to be able to invoke an instance of the tool, while maintaining control over the running tool.

Prior to initiating the tool, the Foreman selects a workspace in which to run the tool. It then initializes this workspace, usually by clearing it of any remaining files. The set of Foremen processes on a TBH are responsible for managing the set of workspaces the TBH has for NSW tool support. The organization and utilization of the workspaces are left completely to the Foreman. However, the WM must be informed as to which workspace a tool has been assigned, so that it can initiate proper file movement into and out of the tool workspace. The host specific parameters describing the workspace (e.g. in TENEX they are a directory name and (if necessary) a password) are returned to the WM as results of the F$BEGINTOOL. If there is currently no available workspace, then the WM request must be rejected. The WM maintains lists of the TBH workspaces which are running tools, and these play an important role in helping a Foreman recover from system crashes without losing user files left in the workspace.

The F$BEGINTOOL message includes flags indicating whether or not to start execution of the tool and whether or not the tool knows about the NSW (new tool/old tool). The message also includes the name of the process on whose behalf the tool was created (usually the FE representing the user), and the process which serves as the FE to the user (if different from the above). An option of the F$BEGINTOOL message is the inclusion of a list of local to the tool file names which are to be directly accessible to the tool. These files are non-NSW files to which the tool should be allowed direct access without any intervention from the Foreman.

2.2 Removing an Instance of a Tool

Throughout the tool session, the Foreman must be receptive to an F$ENDTOOL specifically addressed MSG message from any Works Manager process. The message will contain a reason for ending the tool session. In all cases, the Foreman will return to the caller the cost incurred by the tool which has been run, after the actual termination of the tool. The Foreman will terminate itself after responding to the F$ENDTOOL request, and the association between the tool/Foreman and the NSW system will be broken.

To help Foremen be receptive to the F$ENDTOOL request, the WM will precede a F$ENDTOOL request with an MSG alarm of code=1. Receipt of this alarm code will signal the Foreman of the forthcoming F$ENDTOOL message. Once a Foreman has received an alarm with code=1, it is expected to immediately begin processing its incoming messages, discarding all except the F$ENDTOOL request. Once a Foreman begins processing a F$ENDTOOL message it need not be receptive to any further messages or alarms. A F$ENDTOOL message which was not preceeded by an alarm with code=1 should be processed nonetheless.

The accounting information returned in response to an F$ENDTOOL is a list of accounting data items. The first entry in the list is an integer which is the cost in cents of running the tool for this session. The remaining items in the list are host specific measures of various resources consumed by the tool. Each TBH implementation registers with the NSW which resource measurements it takes and will return to the WM.

(Note: at some future time, it may be advantageous on certain systems to simply halt the tool instead of terminating it, with the possibility of eventually re-initializing the tool as a different instance of the same tool. If such a mode of operation were possible, and if the WM kept track of the tools assigned to each Foreman, then startup costs for tool invocation might be reduced by selecting a Foreman with the appropriate tool already in place. Since this may not be relevant to all hosts, we recommend that part of the F$ENDTOOL message specify whether or not to try to maintain the tool in the inactive state, and that the reply to F$ENDTOOL correspondingly indicate whether or not this was done.)

2.3 External Control of Execution

If possible, the Foreman should be able to handle F$STARTTOOL and F$STOPTOOL messages. These are used to externally control the progress of the tool through its algorithm. F$STARTTOOL can be used to initially start the tool in cases where F$BEGINTOOL did not specify immediate startup. A minimal start/stop facility would provide for suspending the execution of the tool while maintaining its complete current state. The tool would be subsequently continued from the point it was stopped, or it would be aborted. If the tool can be stopped, then it must at least be able to be started (continued) again. The suspending and resuming of tools on certain hosts may not be possible because the host operating system does not support such behavior. In these cases, tool execution will automatically begin with the F$BEGINTOOL message, and F$STARTTOOL and F$STOPTOOL messages will be rejected.

A more extensive start/stop facility encompassing stopping as well as starting through an entry vector is often desirable, and we have made provision for this at the WM command language level. The implementation of this extension is highly desirable where it is possible. We specify the characteristics of the richest facility. It is the responsibility of the Foreman implementation to either reject requests for which it has no mechanism, or to map the request into an alternative well documented in the tool builders guide. The facility the WM supports builds upon the (required of all implementations) F$BEGINTOOL and F$ENDTOOL messages. These requests initially create the tool instance and ultimately (forcefully) cause the tool to immediately cease its existence. These requests roughly translate to the user saying to the WM "runtool" and "aborttool". During the execution of a tool, the user can of course request that the WM stop the tool. To do this the WM sends to the appropriate Foreman a F$STOPTOOL message, indicating the reason for stopping. [Note that all messages originating in the WM and destined for a Foreman of an existing tool are sent using the Sendspecific MSG facility, and can be received using the Receivespecific MSG facility. The responses (if any) are sent directly to the WM which made the request. Requests emanating from the Foreman, however, are sent generically to any WM, with the reply obtained as a message specifically addressed to the appropriate Foreman.] During a period of tool inactivity due to the effect of a F$STOPTOOL, the Foreman still must process MSG messages concerning the tool. It must always be able to process an F$ENDTOOL message, and may have to process any replies to outstanding requests made to other NSW processes (e.g. a request to the WM to retrieve a copy of an NSW file). However, any results to be returned to the tool may have to be queued awaiting the command to continue tool execution.

2.4 Entry Vectors

In a complete Foreman implementation, a stopped tool can be started in a number of ways. We introduce the notion of entry vector to unify the start tool concepts. The WM recognizes several standard entry points for its tools. These are an initial entry point (cold start), a standard re-entry point (warm start), a termination entry point, and a continue from where stopped entry point. (Other system wide entry points will be defined as needed. It may also be possible for tools to implement private entry points invocable via the WM command language.) The meaning of all of the standard entry points is obvious, except perhaps for termination. The intent of the termination entry point is to allow a user to force control to be passed to a final tool cleanup routine, which may also involve saving some of the work of the session. Tools will normally implement commands to do this without WM intervention. However, circumstances may exist (e.g. runaway tool, depletion of resources) where it is useful to force an orderly termination while circumventing the normal tool dispatching. It is expected that the termination sequence implemented by the tool will complete reasonably quickly. If the tool has not signalled its Foreman that tool processing is complete within some host specified time frame, then the Foreman has the right and obligation to forcibly abort the tool execution without further notice.

When installing a tool in the NSW, the tool supplier indicates which entry point functions are available in the tool. This information is maintained by the WM in the interactive tool descriptor, and is used to regulate the type of F$STARTTOOL requests that are sent to the Foreman. In the WM tables the various entry points are statically denoted by a small integer (index). The standard entry points are denoted by the same index for each tool. It is entirely the responsibility of the Foreman and its host operating system to devise a method of converting these indices into actual program entry points (e.g. program counters) for executing the proper function. A Foreman implementation can impose coding standards for these purposes on the NSW tools which want to support the entry point concept.

N.B. When using any form of start/stop facility for tool control, it is the responsibility of the WM command language interpreter to insure the correct sequencing of the start and stop requests. Since in general MSG makes no assurances regarding the order of message delivery, and since WM command language requests are sent generically to any WM, it is most convenient to enforce sequencing at the command language interpreter. All this really means is that to ensure correct behavior, individual tool start and stop requests should not be allowed to be pending simultaneously. A final note on tool control is also in order. The concept of tools controling other tools, as contrasted to the NSW user controling a tool through the WM command language, is currently being defined. We envision the same set of Foreman procedures to be used in tool-tool control. This is discussed further in a subsequent section of this document. However, detailed elaboration of these concepts is postponed to a later date.

2.5 Detailing the Functions

Foreman procedures for tool control: (called by any WM process) The reply to each function invocation includes a "result" variable, which indicates if the operation was successful, and if not supplies a code indicating the reason.

F$BEGINTOOL (program-name, tool-type, entvec, FE-addr, cr-addr, filename-list) -> result, qstart, workspace-descriptor, tool-addr

A request of this type brings the tool instance to life. Program-name is a character string naming the program which forms the body of the tool. Tool-type is a variable indicating the nature of the program as an NSW tool (values defined below). Entvec indicates whether or not to start execution of the tool, and if started, through which entry point. FE-addr and cr-addr are MSG process addresses of the front end process and the creating process respectively. Filename-list is an optionally specified list of non-NSW local files (in the local host syntax) to which the tool is allowed unrestricted access. Some Foreman implementations may not care to protect the access to non-NSW local files. Tools running under such a Foreman would not need a file access list passed to the Foreman at tool initialization time. Tools which do not use non-NSW files would also not need a file list recorded with the WM for the tool initialization procedure. Qstart indicates whether or not the tool execution was begun. A workspace-descriptor is returned to the WM to allow file movement into the tool workspace. Tool-addr is the MSG address of the tool, which is often the same address as the Foreman (see Appendix 2).

F$STARTTOOL (entvec) -> result

The entvec variable indicates how the tool is to be placed back in the executing state: either to be continued from where it was last stopped, or through a specified entry location.

F$STOPTOOL (entvec) -> result

When a WM invokes this function the Foreman stops the execution of the tool and saves its state. We have provided entvec as an argument as a convenience in performing the dual operation of first stopping execution, and then commencing execution elsewhere through an entry point. Attempting to start an already executing tool, or stopping an already stopped tool will elicit an error condition.

F$ENDTOOL (reason, termtype, qmaintain) -> result, accounting-list, qmaintained

Reason is a code indicating why the tool instance is being removed. Termtype indicates the type of file processing required of the Foreman before completing the F$ENDTOOL. This will be further clarified in later parts of the document. Qmaintain is a boolean indicating whether or not the Foreman should attempt to maintain the tool image for use as another instance. Qmaintained is another boolean which the Foreman uses to communicate to the WM whether or not the image has been maintained. Accounting-list indicates the resource cost and utilization for the tool session.

tool-type: value is an index
=1 -> encapsulated tool
=2 -> tool uses NSW calls, does not use MSG
=3 -> tool uses NSW calls, also use MSG facilities

entvec: value is either empty or an index
empty -> do not start tool (if possible)
=0 -> continue from point where stopped (illegal in F$BEGINTOOL)
=1 -> cold start entry point
=2 -> warm start re-entry point
=3 -> termination routine entry point
=4 -> reserved for expansion
=5 -> reserved for expansion
=6 ... -> tool specific entry points

2.6 Voluntary Tool Termination

As mentioned above, the Foreman must provide its tool with a primitive operation for indicating that the tool has completed execution. The HALTME primitive is the means by which a tool voluntarily relinquishes control for the final time. The Foreman may yet have to save files for the tool (see subsequent sections on the file system and encapsulation) before actually removing the job from the NSW domain. Through the termtype parameter of HALTME, the tool can indicate the type of Foreman file processing it expects. The current choices are:

* termtype=1 -> no Foreman file processing

* termtype=2 -> Foreman asks user which files need saving and saves them

* termtype=3 -> Foreman automatically saves latest copy of modified files

After all peripheral operations by the Foreman are complete, the Foreman notifies the WM of the tool completion by calling the WM W$TOOLHALT procedure. The associated parameters of the WM request include the accounting data list describing the tools resource utilization. The tool can be terminated (in the local host sense) any time after it issues the HALTME primitive. The Foreman terminates itself (in the MSG sense) after receiving the response from the WM to its W$TOOLHALT call. A positive response indicates that the association between the tool/Foreman and the NSW has been broken.

Tool primitive operation:

```
    HALTME (termtype) -> never returns to tool
```

WM procedure for HALTME support:

```
    W$TOOLHALT (reason, accounting list) -> result
```

III. Status Probes and Resource Utilization Bounds

The Works Manager, as well as the NSW user through his FE process, may at times wish to closely monitor a tool execution in terms of resource utilization and progress through its algorithm. Toward this end, we specify two functional aspects of a Foreman implementation which can be used to achieve a degree of "tool watching".

Each Foreman must implement an externally invocable function (e.g. requested by a WM process) for probing the status of the tool currently being executed. We have taken the approach of allowing many types of probes. Invoking a status probe is different from invoking almost all other Foreman functions in that it is not done using an MSG message. Rather a status probe takes the form of an MSG alarm signal. The code transmitted as part of the alarm indicates the type of probe. In response to a probe alarm, the Foreman is expected to gather the requisite values and send them via an MSG message (which itself requires nc reply) to the invoking process. Status probes are assigned Foreman alarm codes with values between 10 and 20 (octal). Two probe types are initially identified here, with others added as their need arises. One initial probe (alarm code 10) queries the current state of the tool as a program in execution. In response to a state probe, the Foreman returns the tool's current external NSW state (e.g. running, stopped, running at termination code, etc.), the tool's current internal NSW state (e.g. executing, waiting NSW primitive completion, etc.), and the tool's current local operating system state (e.g. running, blocked for I/O, dismissed, etc., and its program counter). The second initially defined probe (alarm code 11) queries the current state of the tool resource utilization for the session. It returns the accounting list referred to earlier. This includes the cost of running the tool so far, and other resource utilization measures maintained by the host operating system.

Currently there are no immediate implementation plans involving these probes other than as an indicator that the host/Foreman/tool complex is still functioning. At this point it is unclear which processes should be allowed to invoke which status probes, and we may have to specify an (as yet undefined) option for results to be prepared for a human rather than for program processing. Additionally, more thought needs to be applied toward determining a useful set of measures in each probe class. Because of these uncertainties, and because the "still working" function can be fulfilled by responding with an "unimplemented function" response, we postpone the exact specification of the values to be returned in a status probe.

A Foreman should also support the enforcement of resource utilization bounds on the tool it is running. These bounds would specify an approximation to the maximum use of a particular resource or a measure of total resource consumption to which a tool session is limited. Exceeding these bounds would force a cessation of tool execution (i.e. placing the tool in a stopped state) along with the Foreman immediately notifying the Works Manager of the excessive resource utilization. The WM could then apply more resources to the tool session, or cause the tool to execute at its termination sequence, or abort the tool altogether. The initial bounds are passed as part of the F$BEGINTOOL request. We mean these resource bounds to be approximate monitors, and we have no interest in requiring very close scrutiny of the tool execution in order to shut off service the instant the tool exceeds a bound. On the other hand, we would require that the tool be monitored as closely as is reasonable to ensure that it does not consume an arbitrarily large piece of the specified resources (e.g. cpu time).

The WM procedures to support these bounds and the strategies for their use have not as yet been defined. The management tools in the NSW will surely have an impact on the nature of the facility which is actually used. Thus, as with the status probe and other features yet to be discussed, care should be taken not to exclude such behavior, but implementation is not required or yet possible. Future documents will detail the specification of these functions, and we solicit suggestions on their form and use.

Foreman Function:
(invocable via an alarm with code in the range 10-20 octal)

```
    F$STATUSPROBE (type) -> status item list
```

IV. NSW Runtime Environment

The NSW tool environment differs in a few key areas from the environment provided by the host operating system. The NSW has its own means for inter-component communication and for dynamically creating NSW entities, and maintains its own file system. These facilities are in addition to any similar facilities which the host operating system may already provide, and may be used simultaneously if there is no conflict with providing NSW services. The Foreman and other NSW components provide access to the NSW facilities through enhancements to the set of primitive operations available to tools. There are primitive operations for dealing with each of the areas mentioned:

* NSW file system
* NSW process communication
* NSW process creation

These are discussed individually. The common thread is that the operations are provided in operating system-like fashion to tools, with the exact means of invoking a function or obtaining its result/status dependent on the host implementation. We are concerned here with the semantics associated with the primitive calls, and the WM-Foreman message exchanges used in implementing these semantics. It is the Works Manager which actually supports the substance of much of the NSW environment. It is the Foreman which provides the local interface to the NSW facilities.

The functionality of each of the specified primitives must be made available to all tools. However, the exact form of the Foreman calls and returns, and the exact nature of the Foreman tool interface is left to the discretion of the Foreman implementer. Thus where it is deemed desirable, certain calls may be subsumed by a parameterization of other calls, or calls may be coupled to perform multiple functions. These are local optimization issues. We are concerned only that it be possible for each tool to somehow perform all of the operations which we feel to be important in the NSW context.

V. NSW File System

5.1 Two File Spaces

A tool running under the NSW system can independently manipulate items in two distinct file spaces. One file space is the sharable NSW global file space managed by the Works Manager and maintained independently of any tools that manipulate the files. The other space is the non-sharable, temporary workspace (local file space) for the copies of the files in use by a tool during the current tool session. A file entered in the central catalog must have a unique global name, and hence is able to be referenced (though perhaps not accessed) by any tool. A file which exists in the workspace for a tool can be referenced only by the tool operating in that workspace, and the mere existence of such a file may be unknown to other tools and even to the WM. It may also be the case that the name of a file in the workspace is not unique in the NSW file system. There is no conflict as far as the WM is concerned since the workspace file is unknown outside of the tool domain, and the tool itself is provided with a means for resolving any local name conflicts. Explicit name conflict resolution must occur whenever a file is to be entered into the global NSW file space.

5.2 Using Local Workspace

There is a considerable cost associated with inserting a file into the global filespace. The cost of synchronizing local activity with the global directory includes name conflict resolution, file copy (possibly network copy) and delay associated with synchronizing the WM and Foreman. Insertion into the local workspace is immediate. The same cost relationship holds when retrieving files for use by the tool. Therefore, it is often good strategy to build tools which utilize the workspace for storing and retrieving as much as possible, often waiting either until it is explicitly desired to synchronize the file use with other tools or until the end of the tool session before delivering selected files. Delivering files to the global file space at the end of a tool session means that only files which actually need to be permanently saved invoke the large system overhead, while files which do not require permanent name status (e.g. files which are subsequently deleted during the course of a tool session, or files which are only intermediate versions of a particular file) incur a minimal overhead. Savings can also be achieved by delivering multiple files in a single WM request, an obvious optimization if files are batched locally.

5.3 Version Numbers

Another aspect of the local workspace is the automatic use of version numbers to distinguish files of the same name. In essence, version numbering adds another field to "local" file names. The Foreman knows about the use of this part of the name field and supports certain default options for it. A version number is a small integer which gets bumped automatically when creating a file with an already existing (local name space) name and a version number is not otherwise specified. The use of version numbers is not part of the global NSW file space. Therefore the user must disambiguate name conflicts when files are moved from local space to NSW global space. In addition to providing automatic local disambiguation through version numbering, the Foreman must allow a tool to specify version numbers when referencing files in local space, as well as provide reasonable defaults for obtaining latest local copy and creating a newest copy in the absence of specified version numbers. If local files are cached for delivery (highly recommended) the Foreman should provide a means for the user to select from among the "new" NSW files only the ones he actually wants preserved. Suitable defaults are required in the absence of this information (e.g. the highest version number of each different file name is delivered at the end of the tool session).

5.4 Maintaining an LND

In the course of implementing the local workspace concept, the Foreman is required to maintain a local name dictionary (LND) for the tool it is running. The LND is used to specify the relationship between the NSW file name and the name of the file (in local operating system terms) which represents the local copy of that NSW file. Other information about the files which are created and maintained during a tool session is also appropriate for the LND. This includes information about version numbers, indications of whether a file has been modified since the copy was obtained (and therefore may need to be delivered), and short abbreviated strings which the NSW user or tool uses to refer to the NSW file. Some of the primitive operations provided to the tool are expressly for the purpose of manipulating the contents of the LND, and hence indirectly manipulate the local workspace. It is imperative that the LND for each tool instance be kept in a "crash-proof" manner (or as near to this as is reasonable and possible), so as to make feasible a recovery procedure in the event of a host system crash. Maintaining the LND in a carefully maintained and identifiable file on the local file system is one technique for achieving this. After a system crash which is not so catastrophic as to destroy the file system also, it would be possible to run a scavenger program in the tool workspace to retrieve the appropriate LND and save some of the results of the tool session. When the host again becomes available, it may also be possible to re-start (or continue) the use of the tool in the same workspace and working with the saved LND. These issues are further discussed later in this document. In any event, the Foreman should attempt to insure that the effect of the host system crash is no worse for the NSW tool user than for the direct (non-NSW) users of that system.

5.5 File Names in NSW

The general syntax and semantics associated with file names in the NSW are described elsewhere. Here we are concerned with the impact of the Foreman and local workspace concepts on the use of NSW file names. The impact is two-fold: first in the conventions used by tools in providing names as parameters for file system operations, and second as extensions to the name syntax to provide for the manipulation of files in the local workspace.

To discuss file names as parameters to file system operations we must first describe certain aspects of the central NSW filing system as implemented via Works Manager procedures. Generally, the WM file system procedure for retrieving a copy of a file requires only a partial name (filespec) to specify the NSW file for the operation. Specifically, only enough of the name need be specified to disambiguate it. Thus we have the concept of files being retrieved (and saved) using abbreviated names. For example, WALDO.AUTHOR.TEXT might be addressed simply as WALDO. Additionally, when a file name provided to the WM is ambiguous (for retrieval) or already exists (for delivery), the WM often negotiates directly with the user to clear up any uncertainties. This is done only when requested by the tool. Because of these features, most calls involving file names return values which indicate the full NSW name of the file which was actually operated on, as well as any change in the filespec for the file as determined from the interaction with the user. (If the user dialog results in a new filespec, the user will presumably use this new name in future references to the file.) It is a Foreman responsibility to keep an up to date LND reflecting the latest information about NSW file names, as well as to make these names available to the tools which may be unaware of the change or clarification from the user. The names can be provided to the tools either by returning their values directly as part of the tool file operation, or (recommended) by providing functions by which tools can retrieve the names when they are needed. Since the names are kept in the LND anyway, such an operation is rather simple.

To insure that a tool can achieve a maximum utility out of the separation of global and local file spaces and the properties of both, the file manipulation primitives each imply an explicit domain (i.e. global or local space). This allows the tool to directly control the spaces individually in the manner most appropriate to its particular purpose. Primitives dealing with workspace files also provide for the explicit selection of a particular version of a file, to enable a tool to override any default assumptions. Supporting a reasonable set of default parameters is encouraged, provided the defaults can be overriden where appropriate.

5.6 File Semaphores

Associated with each file in the NSW (global) file space is a semaphore. This semaphore can be set by a tool on behalf of a user (who intends and is able to modify the file) in order to warn other potential users that the file may be undergoing change. In general, this semaphore serves as a loose lock- the NSW system will only warn other users, not restrict their access. Under some circumstances, however, users will be prevented from accessing a file with the semaphore set. Note that the general looseness of the lock does not cause a multiple writers problem. NSW files are not directly modified by tools. Only local workspace copies of NSW files are directly modified. The NSW file is not modified until it is explicitly replaced. Thus a user can obtain an internally consistent copy of an NSW file with semaphore set, since it is only a workspace copy of that NSW file which is actually undergoing modification. The user is warned by the set semaphore that the modification is taking place (by some other user), and that at some time in the future, the NSW file may be replaced by an altered, updated version.

Tools may be divided into two classes: tools which explicitly use semaphores and tools which do not. In the first class go tools which note when a user performs a file modifying action and which are then willing to request the setting of a semaphore. Tools of this class may request that a semaphore be set when a copy is obtained of an NSW file. If this request is made, then the Works Manager presumes that the tool does not want access unless the semaphore can be set. If it is already set (by some other user) then the access request is blocked. In all other cases of copy access, the requester of a file is merely informed that the semaphore is set (and by whom- project and node-name). Delete access is always blocked by a set semaphore.

Whenever a tool of the first class (explicit semaphore users) requests a file, it may also request the setting of the semaphore. As noted above, inability to set a semaphore blocks the access request. Subsequently, a tool of this class may request the setting of the semaphore for a previously obtained file. The semaphore may be read or unset at any time, either by the tool or directly (via a WM command) by the user. If the semaphore has been explicitly requested, then it may remain set after the end of the use of the tool.

Whenever a tool of the second class (non-semaphore user) requests a file, the WM automatically tries to set the semaphore. Failure to do so does not block the access; the user is merely warned that the semaphore is already set. Again, the semaphore may be read or unset at any time. The semaphore is automatically unset when use of the tool is ended.

In all cases, a semaphore is unset whenever an NSW file is replaced (or, obviously, deleted). The tool primitives to be implemented for interfacing to the semaphore procedures of the WM are listed below. See the Works Manager Procedures document for details of the implementation within the WM.

```
SETSEMAPHORE (filespec, qhelp) -> result, NSW filename
UNSETSEMAPHORE (filespec, qhelp) -> result, NSW filename
READSEMAPHORE (filespec, qhelp) -> result, NSW filename
```

5.7 File Manipulation Primitives

A tool is provided with distinct sets of primitive operations to individually manipulate the NSW global filespace and the local workspace. An additional set of operations is provided for moving file copies between the two spaces. In this section, we introduce the major file manipulation primitives in each of the three sets.

A tool is provided with primitives for deleting, renaming and copying files within the global NSW namespace. The WM implements procedures which actually perform the DELETEGLOBAL, RENAMEGLOBAL and COPYGLOBAL operations within the global space, so these primitives are merely a packaging operation for calls on those procedures.

Two tool primitives (GET and PUT) are provided which are normally used to relate the files spaces in an automatic fashion. These provide for obtaining a local workspace copy of a global space NSW file (GET), and for depositing a local workspace file as a global space NSW file (PUT).

To support local workspace file access, the Foreman provides primitives for deleting, renaming and copying workspace files, and OPEN and CLOSE primitive operations. DELETELOCAL, RENAMELOCAL, and COPYLOCAL are all implemented totally within the Foreman, and merely result in changes to the LND. The OPEN primitive is used, with appropriate parameters, to gain access to a local workspace file, or to create a new workspace file in cases where one does not already exist. The CLOSE primitive is used to signal the Foreman that the file access is complete, and that the Foreman should assume responsibility for that version of the file. It is the intent that OPEN should prepare a local workspace file for direct access and modification by the tool, and generally to return to the tool a handle on the file for such access. The type of handle, as well as the types of file data manipulations which are permitted, are local host operating system dependent, based on the file system which underlies the workspace implementation. One aspect of the CLOSE primitive is the invalidation of such a handle so that the Foreman can maintain a consistent copy of the file (via the LND) for possible introduction into the global file catalog. After executing a CLOSE operation on a file, the file data is not accessible to the tool unless it executes another OPEN.

As a general scenario, a tool GETs a local workspace copy of the global space NSW file it wishes to access. The workspace OPEN provides for direct access to the file data. The file actually accessed is always a local workspace file. It may be a copy of a global space file, or it may be a newly created, currently empty local workspace file, or it may be a previously referenced local workspace file which was originally one of the above. We assume that the local host system provides the actual data manipulation primitives for local workspace files. The tool ultimately CLOSEs the file and may subsequently repeat the OPEN-CLOSE sequence some number of times during the tool session. These operations affect only local workspace file images, and new versions of the original file copy may be created. Ultimately the tool decides that some version(s) of the workspace file should be placed into the global catalog, and it instructs the Foreman to do so using the PUT operation. Although these operations give the tool complete control over all phases of its file system interactions, Foreman implementations can and perhaps should often provide simplified means for performing common file functions. As examples, we could consider a Foreman which provides a GET which has the option of automatically opening the retrieved copy of the file, or an OPEN which searches the local space and if unsuccessful tries to GET a copy of the file, or automatic PUTting of the highest new version of each different workspace file upon tool termination. Any such local options will be clearly noted in the host specific tool builders guide.

5.8 Specification of the File System Primitives

In specifying the parameters of the main file manipulation primitives, there are many common arguments. Some are described here as a general introduction to the primitive descriptions.

NSW-filename: The NSW-filename is the full identification of a file in the NSW file system. This is generally a rather long string of text. However, a user will never have to type in a full filename. Instead, he will use either a "filespec" or an "entry-name" (defined below) depending on the intended use of the file. A full NSW-filename consists of two parts: the name part and the attribute part, separated by a slash (/). The name part is a sequence of name components, separated by periods (.). The attribute part is a list of attributes separated by semi-colons (;). (For a more complete description of NSW-filename and attributes, see the document Works Manager Procedures.)

filespec: This is basically the supplied identifier for an NSW filename. It is an abbreviated form of an NSW-filename, used in contexts where the name of an existing file is required. A filespec need contain only enough parts of the NSW-filename to unambiguously denote the file. Unless changed by the tool, a workspace file copy and all its derivative versions will be referred to for the duration of the tool session by the filespec. A filespec may also contain file attributes as part of the name. (For a more complete description of filespec, see the document Works Manager Procedures.)

entry-name: An entry-name is an abbreviated form of an NSW-filename used in contexts where a new filename is to be created. As described in Works Manager Procedures, the contents of the user's enter scope is prefixed to the entry-name by the WM when a file is delivered. Aside from this abbreviation, however, the user (or tool) must specify the entire name component of the file. (For a more complete description of entry-name, see the document Works Manager Procedures.)

version #: When the local workspace is searched, the version number parameter guides the selection of a particular instance of the files associated with the filespec. Version number is either null (default) or is a decimal integer in the range 1 to 32. Special indicators exist for referencing the highest version, one more than the current highest version, and the lowest version of a filespec. The default value of version number is different for the various primitives and is given along with each primitive description. In general though, defaulted version numbers use the highest version for file access and creates a new highest version for file deposit. When searching the global space, any version number information is ignored.

qhelp: This argument has three possible values and conveys to the WM how the tool would like filename conflicts handled. It is used in conjunction with the various file system primitives. The possible values and their meanings are:

* qhelp=0 -> allow the user to supply help, through a help call on his FE process

* qhelp=1 ->allow the tool to provide help through a help call back to the tool

* qhelp=2 -> do not provide any help but instead report a failure on filename conflict, indicating this as the reason

The qhelp parameter applies only to calls involving the WM and the global NSW file space. Primitives dealing with workspace files need not support these help notions. Version number defaulting provides a form of automatic disambiguation for local space files. The Foreman must not allow the existence of different workspace files named with the same filespec (although different versions of the same file are obviously permitted). The WM interactive tool descriptor provides for a static default of the qhelp parameter, if the tool builder so desires. For these tools, the value of qhelp is obtained (each time) from the WM tables, regardless of the value passed by the Foreman. This is especially useful for encapsulated tools.

qreplace: This argument is a boolean, and it indicates whether or not a file being placed in the global space should force replacement of an existing file of the same NSW name. Qreplace with value true means that such replacement should be automatic, with the old file no longer accessible. Qreplace with value false means that replacement is not automatic and that the WM should revert to the qhelp variable to determine how to deal with the conflict.

success/failure code: An integer value representing the success/failure code for the operation is always returned as a result of each primitive. Each individual primitive has associated with it a set of interpretations of these integers. This code is always returned as a primitive result, but it will not be explicitly shown as a return value in the following primitive descriptions.

The description following each primitive operation reflects the nature of the operation as seen by the tool. In this section we limit ourselves to a description of the tool primitive (i.e. what does the primitive do?), leaving implementation considerations for the next section.

The tool has unrestricted access to all of the files within its workspace. Once a copy of a global space file has been obtained, all references to the local copy are permitted. The global space, however, is tightly controlled, based on the capabilities of the user and the tool he is using. Each operation involving a global NSW file undergoes strict access checking by the WM before it can be accepted. This is not of concern to the Foreman implementation, since the WM provides all of the access checks. The types of checks are mentioned here only to completely describe the primitives that the tool uses. The WM file system and its access controls are described in the Works Manager Procedures document.

5.8.1 Global NSW Filespace Primitives

1. DELETEGLOBAL (filespec, qhelp) -> NSW-filename

This is the primitive a tool uses to delete an existing file from the global NSW file space. Global space deletion cannot take place until the WM verifies that filespec designates a unique file to which the user has delete access. This access is blocked by a set semaphore. Assistance is obtained as indicated by qhelp. Once a file has been deleted, it will no longer be accessible for GET, COPYGLOBAL, RENAMEGLOBAL, etc. The actual full NSW filename of the deleted file is returned to the tool after a successful DELETEGLOBAL operation.

2. RENAMEGLOBAL (filespec, entry-name, qhelp, qreplace) -> src-NSW-filename, dst-NSW-filename

A tool uses this primitive to change the name of an existing global space NSW file. It renames one global space file to be another global space file. The WM verifies that filespec designates a unique file to which the user has delete access. Enter access is also required for generating the new file name. The new file acquires any tool supplied attributes of the old file. A set file semaphore blocks a RENAME operation. Qhelp and qreplace are used according to their definition. Both source and destination NSW filenames are returned to inform the tool of the operation which actually took place.

3. COPYGLOBAL (filespec, entry-name, qhelp, qreplace) -> src-NSW-filename, dst-NSW-filename

A tool uses this primitive to create a new global NSW file which is a copy of an existing global space NSW file. It is similar to RENAMEGLOBAL with the exception that on completion both the source and destination files exist. For global space copy, enter access is required as well as delete access if copying to an already existing file name. Qreplace is a boolean which when true causes file replacement to be the default on collision of file names. Qreplace with value false means that either help must be obtained or the operation must fail. Both the source and destination full NSW-filenames actually used to complete the COPYGLOBAL are returned for the information of the tool.

5.8.2 Primitives for File Movement Between Spaces

4. GET (filespec, input-attribute-code, qset, qhelp) -> NSW-filename, new-filespec (only if changed), version #

A tool uses this primitive to cause a copy of the global space file denoted by filespec and having the attributes specified by input-attribute-code to be moved into the tool workspace. This working copy can then be manipulated by the tool using workspace file access primitives. When GETting a copy of a file obtained from the central catalog, the WM verifies that the user has copy access to the file, and that the file has the input-attributes specified by the tool. If these conditions are met, the WM initiates the proper actions to have a copy of the file moved into the tool workspace. This file movement may involve a network file transfer. When calling for a file transfer, the WM also insures that any file conversions which are necessary and possible are indeed performed. File conversions are based on the current state of the file and the intended use of the file by the tool (see The File Package document). Qset indicates whether or not the tool desires to set the semaphore associated with the original copy of the file. Note that in the event that the tool does not choose to utilize semaphores (this is indicated in its static tool descriptor) then the WM may automatically set the semaphore regardless of the value of qset. Qhelp indicates how the tool wishes to handle name ambiguity. The full NSW-filename of the file actually copied into the workspace is returned, as is any new filespec for this file (possibly obtained via user help). The version number of the workspace copy is also returned for the information of the tool. GETting a copy of a file for which the workspace already has the matching filespec and NSW-filename causes a new highest version to be created. GETting a copy of a file for which the workspace already has a matching filespec with a different NSW-filename will cause an error return to the tool.

5. PUT (filespec, version #, entry-name, output-attribute-code, qreplace, qhelp) -> NSW-filename

A tool uses this primitive to place a copy of a workspace file into the global NSW catalog. The file is identified by filespec and version number. Entry-name is the full NSW-filename (less any defaulted Entry scope) the tool wishes the file to have. If not specified, entry-name defaults to the name part of the full NSW-filename contained in the LND entry for the filespec (this is usually the same name as the one returned from the GET operation). If the LND has no NSW-filename and entry-name is not specified, then the Foreman returns failure to the tool. The WM requires that the user have enter access in order to deliver new files into the global space. The output-attribute-code is a tool dependent code denoting an attribute which should be associated with the file. The WM will convert these codes to textual attributes which become part of the full NSW-filename. Qhelp guides the WM in seeking help with filename conflicts, and qreplace indicates whether the tool desires that the current file replace any file which may already exist with the same name. The full NSW-filename of the file as it is put into the global catalog is returned to the tool. A copy of the file also remains in the workspace, and can be referenced again using any workspace file manipulation primitive.

5.8.3 Primitives for Workspace File Manipulation

6. OPEN (filespec, version #, new-file-flag, old-file-only-flag, type-of-access) -> file-handle

The tool uses the OPEN primitive when it wants to actively access file data in a workspace NSW file, or to create a new workspace NSW file. A successful OPEN returns a handle for the referenced file. The handle is intended to be used when subsequent manipulations of the file data are requested. The nature of the handle, as well as the primitive operations available for the actual data manipulation are host dependent, based on the existing host file system, and are beyond the scope of this document. The handle is necessary since tools use NSW syntax for dealing with NSW domain files, and for the most part remain ignorant of the intermediate representation of the file in the local host file syntax. However, a Foreman implementation is not forbidden from using the local host syntax for the file as the handle returned from the OPEN, although this is not recommended. The file handle is also used to query the Foreman regarding any NSW information the Foreman maintains about the file, including the full NSW filename, known attributes, etc., should such a primitive be implemented.

If the new-file flag is set, a new local workspace file is created using filespec and version number (default is next higher version). The only failure for creating new files, other than failures due to the nature of the specific workspace implementation, is when specifying a version number of a filespec which already has such a version.

If the old-file-only flag is set, then success can be returned only if an existing file adhering to the filespec,version specification is found. If neither the new file flag nor the old file flag is set, then a failure to find an existing workspace file results instead in creating a new local file referenced by filespec.

The type-of-access parameter is optionally specified by the tool to indicate more precisely the type of file access it requires (e.g. read, write, read&write). The default for type-of-access is read & write. A Foreman may find the type-of-access information useful in determining whether a file is being modified (and may need to be delivered back into the global space), and in utilizing the structure of the underlying file system.

The search for an existing file matching filespec is within the local workspace only. Version number defaults to the highest existing version (except for new file as outlined above).

7. CLOSE (handle, output-attribute code, qdisp) ->

The tool uses this primitive to indicate that it has completed accessing the file denoted by handle and that the system should now assume responsibility for it. In addition, using qdisp the tool can guide the Foreman as to the ultimate disposition of the file i.e. whether it needs to be placed as a global NSW file (qdisp=true), thereby becoming referenceable by other NSW tools; or whether it can remain a workspace file until the end of the session or until such time as the tool instructs the Foreman to do otherwise (qdisp=false). The default value of qdisp is true, i.e. deliver the file at the end of the session. Completion of the CLOSE invalidates the handle for the file, and further access to the file must be preceeded by another OPEN.

The output attribute code is a tool dependent code denoting an attribute(s) which should be associated with the file. When the file is delivered to the global NSW space, the WM will convert these codes to textual attributes which become part of the full NSW filename.

8. DELETELOCAL (filespec, version #) -> version of deleted file

This is the primitive a tool uses to delete an existing file from its workspace. For DELETELOCAL only the workspace is searched for the matching filespec. The default version number is the lowest numbered version. Once a file has been deleted, it will no longer be accessible with OPEN, PUT, etc. The version number of the file actually deleted is returned to the tool on a successful deletion.

9. RENAMELOCAL (from-filespec, from-version #, to-filespec, to-version #) -> from-version #, to-version #

A tool uses this primitive to change the name of an existing workspace file. It renames one workspace file to be another workspace file. The new file acquires any tool supplied attributes of the old file. For RENAMELOCAL, from-version #

defaults to the highest existing version and to-version # defaults to a new highest version for the file. The version number of the files actually operated on are returned to the tool.

10. COPYLOCAL (from-filespec, from version #, to-filespec, to-version #) -> from-version #, to-version #

A tool uses this primitive to create a new workspace file which is a copy of an existing workspace file. Both the "from" and "to" files exist in the workspace on successful completion. Note carefully that COPYLOCAL merely makes a copy of the file. It does not provide access to the file data. In general, since the actual local host name of the file remains unknown to the tool and no handle for it is provided through COPY, accessing the file requires an OPEN primitive. The from-filespec can not normally be defaulted, but the to-filespec defaults to that selected for the from-filespec. Default versions are highest version and next higher version for from-version and to-version respectively. The version numbers of both the "from" and "to" files are returned to the tool.

5.9 Other File Related Primitives

There are a few other file related primitives which are thought to be needed but not necessarily for the current set of tools in the initial configurations.

5.9.1 Global Space Primitives

11. WARRANT (filespec, attribute code) -> new NSW-filename

This primitive is used by a tool to assign attributes to a global space NSW file. (Recall that attributes can also be assigned to an open file at the time it is CLOSEd, and also when it is PUT into the global catalog. These are probably the most prevalent means for assigning attributes to files). The attribute code is a tool specific indicator for textual attributes which become part of the file name. The new full NSW-filename is returned to the tool, since the result of a WARRANT may actually change the filename. Filespec must uniquely identify an NSW file. Help is not provided, since only tools (i.e. not users) can assign attributes to a file. The warrant capability is not yet supported by the WM and therefore need not now be supported by the Foreman.

[For completeness, we refer the reader to the previously mentioned semaphore related operations, which are also global space primitives.]

5.9.2 Local Space Primitives

12. GETFILEDESCRIPTOR (local filespec or handle, version #, data fields) -> data structure with specified items

(A primitive of this type is an optional implementation item). This primitive is used to view the information associated with a workspace file through its LND. Typical data fields will include: full NSW-filename, file attributes, existing versions, etc.

13. CHANGEFILEDESCRIPTOR (local filespec or handle, version #, data structure with changed items) -> change outcome indicators

(A primitive of this type is an optional implementation item). This primitive is used to change the LND information associated with local workspace files. Some LND information may not be subject to change. The exact nature of the information kept in the LND will be implementation dependent.

5.9.3 File Movement Into and Out Of the NSW System

The following four primitives are used essentially to move files into and out of NSW controlled spaces, either the global NSW filespace or the tool workspace. The primitives serve as interfaces to Works Manager facilities of the same name. READDEVICE and WRITEDEVICE are used to move copies of non-NSW files into a tool workspace, and to move copies of workspace files into non-NSW controlled space. IMPORT and EXPORT perform the same functions using the global NSW filespace as its base of operation. By non-NSW file space we mean not only space on file oriented devices, but also physical devices such as card readers, line printers, magnetic tape, etc. A user profile guides the default locations for the various physical devices. That is, a tool might request that a particular file be written (WRITEDEVICE) to the LPT (lineprinter). The user profile would indicate which lineprinter was local to the user, and perform the transfer. The details of using the user profile are currently being worked out.

14. EXPORT (filespec, external-name, password, qhelp) -> NSW-filename

EXPORT copies a global space NSW file to a non-NSW destination. EXPORT verifies COPY access and sends a copy of the source file to the location designated by external-name. An external-name is either an ARPANET pathname or a device pathname. Password is a string which is used for gaining access to the external directory, device, etc. The full NSW-filename of the file actually EXPORTED out of the NSW file system is returned for the information of the tool.

15. IMPORT (external-name, password, entry-name, qhelp) -> NSW-filename

IMPORT is the inverse of EXPORT, i.e. bringing a non-NSW file into the global filespace.

16. READDEVICE (external, password, filespec, version #) -> version #

READDEVICE is used by tools to input from sources outside the NSW without making a global space file. The file is placed directly in the tool workspace. Version # defaults to a new highest version. The actual version number of the created file is returned to the tool. When (if) the file is placed in the global file space, it must be given a full NSW-filename.

17. WRITEDEVICE (filespec, version #, external-name, password) -> version #

WRITEDEVICE is the inverse of READDEVICE i.e. copying a workspace file directly to a source outside the NSW domain. Version # defaults to the highest existing version. The version number of the file actually transferred is returned to the tool.

[Only IMPORT and EXPORT are available for tool invocation in the current version of the WM.]

5.10 Implementation of the File Primitives

The WM has procedures that can be invoked by the Foreman to implement the global space file manipulation operations. There are procedures for deleting, renaming, and copying global space files. These procedures and their call/return sequences are described in the Works Manager Procedures document. Each procedure is invoked by sending a generically addressed message to a Works Manager process. Every procedure call generates a reply which can be obtained using the ReceiveSpecific MSG primitive. Replies to multiple outstanding procedure call messages can be distinguished through the conventional use of transaction IDs. These IDs are generated by the invoking process (Foreman) and are included in the message specifying the procedure call. The recipient of the message (a WM process) includes the transaction ID of the call in any reply that it generates. The NSW message transmission conventions (see Postel's note of 10 March 1976) also include indicators of whether a message is a new request or a reply to a previous request. This enables the Foreman to distinguish replies for its WM requests (e.g. W$DELETE) from WM commands regarding the tool (e.g. F$STOPTOOL), since both types of messages are received using the same ReceiveSpecific MSG primitive.

For GETting a local workspace copy of a global space file, the Works Manager's W$OPEN procedure is invoked. The local host syntax file name (of the new workspace file) which the WM returns is used as part of the basis of a new LND entry reflecting the NSW name given to the copy. For PUTting a file into the NSW global space, the Foreman merely invokes the Works Manager's W$DELIVER procedure regarding the local host file indicated by the LND entry associated with the workspace filespec.

For local workspace file delete, rename and copy the obvious LND manipulations are performed. The local host operating system will usually provide help in actually deleting the files, should this be desirable. If not, and also in the case of RENAMELOCAL, merely changing the contents of the appropriate LND entry is sufficient, since the tool does not deal with host syntax file names anyway. The OPEN and CLOSE primitives are implemented entirely within the Foreman to perform the function of relating the NSW file syntax and conventions to the underlying host file system.

The Foreman is not expected to implement controls over the type of access a tool has to workspace file copies since there is no NSW concept of file write access, append access, etc. The NSW is based on getting xerox copies of files, performing arbitrary operations on the copies, and then trying to deposit the altered copies back in the global space. It is the act of obtaining a copy (copy access) and the act of placing a new file (enter and possibly delete access) in the system that require access control. (However, since the host file system may require more specific access type information, the implementation of a Foreman for a particular host may require additional parameters indicating the type of access a tool needs to the particular file (i.e. the type-of-access parameter of the OPEN primitive). Whether or not this is included, at file CLOSE time, it is the responsibility of the Foreman to attempt to determine whether or not a file has been modified, and may therefore be a candidate for re-delivery into the global catalog. Modified files should be so marked within the LND as an aid in post-tool delivery decisions.

5.11 Extension of the File Name Syntax

In the implementation of primitives which refer to a tool user's files, it is often useful to have the system itself (in this case the Foreman) gather from the user the strings for identifying files. An example of such a facility is the GTJFN (Get JFN) system call in the TENEX operating system, where as an option, the program can defer the actual accumulation of the filename string to the operating system. The alternative is to have each tool gather its own filenames by using available communication facilities. For tools that utilize direct channel communication with the user, having the option of specifying the connection to the user instead of a filespec, and letting the Foreman gather the filename string can lead to a much simplified tool implementation. It is recommended that Foreman implementers consider such an interface to their file system primitives.

However, whether the Foreman or the tool gathers the filenames, the user is often the ultimate source of the parameters supplied with the file system operations and as such, the NSW user must be provided with a way to syntacticly specify the exact file on which to operate. That is, the user level NSW file syntax must at least include an option for specifying a particular version from a set of workspace files. If a tool does not provide separate user commands for operating on local and global files, then it may also be necessary to syntactically specify the space to which a filespec refers. We think it important to present these features uniformly to the user, independent of the tool/Foreman he is currently using. In that regard, we are now specifying a syntactic extension to the NSW filename, which can be used by NSW tool users to explicitly specify a version of a particular file. Further extensions delimiting the domain of a filespec may also become appropriate. We emphasize, however, that these extensions are usable only within tools and Foremen, and have no meaning whatsoever at the WM command language level. A user must understand the local workspace concept and when it applies to grasp the meaning of the syntactic extensions.

The syntactic extension consists simply in adding a field to the end of the NSW filename syntax. This optionally specified field is to be delimited at the beginning by a semi-colon (";") character. Following the ";" can currently only be a decimal integer between 1 and 32. This indicates a particular version of a file. By its very nature, a file specified with a version number must be a workspace file, since version numbers are not supported at the global space level. Other extensions will be defined as necessary. Filenames which do not include any extensions (currently a version number is the only possible extension) will take the normal default for the particular operation.

example:

```
   WALDO.GEORGE.TXT;23
```

This file name selects version 23 (only) of WALDO.GEORGE.TXT in the local workspace for use or creation depending on the context in which it is used.

The determination of the version can be derived from either the parameters associated with a call (i.e. version #) or explicitly from the syntax of the filespec provided. Filespec syntax takes precedence over tool parameters in the event of conflicts, since we assume the user to be responsible for most syntax related directives.

VI. Tool to Front End Communication

The NSW user accesses the NSW system through a Front End process. For those tools that require direct user involvement, the Foreman and the Front End must cooperate to provide channels for the communication. We are firmly committed to providing tools with the ability to utilize MSG for both message type communication and direct connections with the FE. The FE could interpret and package user input and transport the pertinent data to the tool in a network MSG message. The tool to FE communication could be handled in an analogous fashion. Another approach to tool/FE communication is through the use of direct network connections. This would typically take the form of an ARPANET telnet connection pair from the FE directly to the tool. The decision as to which type of communication facility a tool uses is left entirely to the tool builder. The extent and type of user interaction which the tool supports, as well as the possibility of additional burden on the FE system must be weighed in selecting a mode for tool communication. Using the techniques outlined in the MSG document addition NSW Note #11 (and included as Appendix 2 of this document) we will support tool communication with the FE using direct (but controlled) tool access to both the message and connection oriented MSG facilities. A tool will be able to selectively use messages, or sets of connections, or both, depending upon the tool circumstances. However, again letting immediate necessity drive our initial efforts, we find that the initial tools are not written using a message type FE interface. Rather, they utilize a terminal oriented interface, best served by a direct connection from the tool to the FE (and hence the user). Because of this, we temporarily defer extensive details of the tool-FE message interface. These details will be of primary concern immediately after the initial Formen implementations are complete. We do require however, that the Foreman support direct FE to tool connections as an immediate objective. This does not require any of the modifications mentioned in Note #11, and hence is in line with the short term implementation plan for all NSW components.

The Foreman initially received the MSG process name of the FE process servicing its tool (see description of F$BEGINTOOL request). Based on this information, the Foreman is required to implement a CONNECTION-TO-FE primitive operation for its tool. To establish the (Telnet) communication path between the FE and the tool, the Foreman sends an MSG message to the designated FE process. The message is a request to exchange MSG connection operations, and indicates that the connection should be of type telnet. After sending the request, the Foreman immediately issues its MSG Openconn primitive directed toward the FE process. If the Openconn succeeds, the handle for the connection(s) is returned to the tool as the response to the CONNECTION-TO-FE primitive. If the Openconn fails after a sufficiently long timeout and retry period, then the Foreman reports failure to the tool. The MSG message sent to the FE process requesting the connection requires no acknowledgement. The completing of the connection serves as a positive acknowledgement to the request.

In cases where the Foreman knows that the tool requires a direct FE connection (e.g. encapsulated tool), the implementation may be such that the Foreman acts to create the connection without requiring the tool to request it. However it is accomplished, the initial Foreman requirement is that each tool be provided with a means of using a direct telnet connection to its FE process. The exact nature of the FE support for tool connections is detailed in the forthcoming document describing a minimal Front End.

VII. Tool to Tool Invocation and Communication

7.1 Present State

The mechanisms for tools invoking other tools or interacting with other existing, background tools, and then for tool-to-tool communication certainly constitute a part of the abstract tool environment. However, no tool from the set of initially anticipated tools needs to use such facilities. Therefore, we are postponing the precise description of the mechanisms provided to tool builders for dynamically creating other NSW entities and communicating/synchronizing with them. At this point however, a rough sketch of the planned me hanisms and a possible implementation strategy can be given. It must be emphasized that much of the content of this section is still in the design stage, and is presented here only to give a more complete picture of a future direction. The emphasis placed on these areas is dependent on the nature of the tools which will populate the NSW, and on whether or not people are willing to customize their tools for the NSW. To even allow the possibility of extensive customization, we are presenting the concepts surrounding these other aspects of the tool environment. It is difficult to judge the impact of these extensions in the absence of tool candidates which need to make use of them. However, we will pursue the refinement of some of the tool-to-tool concepts so that as tools emerge which require such facilities (as they surely will), we will have a cohesive approach for handling them. Implementation may await an expected use. To this end, we invite comments and suggestions on these more complex uses of the NSW.

7.2 Emerging Tool-Tool Concepts

As in the file system operations, the WM and the Foreman in combination are responsible for the dynamic creation and communication aspects of the tool virtual machine, with a large assist from MSG. A general overview of the supported facilities is that there is a variant of the Works Manager's W$RUNTOOL procedure which can be invoked via the Foreman by a tool to create a new instance of another tool. Each tool will be able to use MSG facilities for sending and receiving specifically addressed messages, sending and receiving alarms, and setting up and taking down direct connections, all to a selected list of conversants which includes any tools it has started. Some of the ancillary features of MSG are expected to be included in the tool domain (e.g. Rescinding MSG primitives, Resynchronization with a correspondent). The direct use of MSG by tools is based on the concepts outlined in Appendix 2. The tool will also be provided with primitives for locating service facilities which are implemented as tools, but which do not dynamically become part of the initiating tool's job, as is the case with tool-to-tool creation. With proper verification, the tool can then engage in message and/or connection oriented exchanges with the service tools.

For dynamic tool creation as well as for trying to locate and utilize a background service tool, we provide tool primitives which are fielded by the Foreman. The WM implements procedures which perform the access checking as well as establishing new components where needed using MSG facilities, and returns the pertinent information to the initiating Foreman. The information returned includes the MSG process address of the new tool. The initiating Foreman then manipulates its tool's environment using MSG primitives to allow message and/or connection type of communication between the tools. The initiating tool can specify the MSG process name of a process in its family tree which is to serve as the FE process to the new tool. The Foreman of the newly created tool receives the address of the creating tool as well as the process which is to serve as the tool FE (if any) and adjusts its tool's environment to facilitate communication with these processes. In addition, we envision providing primitives with which the creating tool can control the progress of the new tool and terminate it, in much the same fashion as the user can control a tool through the WM command language.

Once the tools are in communication, they can pass around the names of other tool instances that they know about. We perceive a Foreman call by which a tool can ask that "communication to process xxx" be allowed. The Foreman would try to get the WM to consent to the pair as legitimate conversational partners. If they are, the appropriate Foremen are notified to adjust the MSG environment for their tools, and communication can proceed without further Foreman intervention. How the WM decides if two tools should be allowed to communicate is an aspect of the tool-tool model still undergoing investigation. Another very important aspect of the tool-tool problem, which also has no immediate solution, is handling the situations in which system failures cause breaks in the process trees.

VIII. Tool Encapsulation

The initial TENEX approach to integrating tools into the NSW was through an encapsulation technique. This approach has proven very successful, and we, therefore, feel that each Foreman implementation should consider a similar facility.

In general terms, NSW encapsulation implies the automatic trapping and translation of local host operating system calls into calls meaningful in the NSW system. Any trapping and translation is done within the Foreman process. Using an encapsulation technique, we take programs which are written exclusively for the local host operating system execution environment, and with little or no modification execute them as NSW tools. This is possible only because of the similarity, in many aspects, of the NSW system to a conventional single host operating system. As an example, when an encapsulated tool issues a local system primitive to gain access to a file, the Foreman could get control and translate the request into one which provides access to an NSW file. This assumes that the "old style tool" is somehow capable of handling the NSW filename syntax within the local host file manipulation primitives. In TENEX, for example, this is often be very easy since the tool will frequently allow the "system" to gather the filename from the user. In TENEX encapsulation, the Foreman is interposed between the tool and the operating system only for selected system calls. With its intimate knowledge of both the local system primitives and the NSW system structure, the Foreman gathers the NSW filename and ensures that the tool utilizes NSW files. Using both local host facilities and facilities supported by other NSW components (e.g. WM), the Foreman "implements" the local host file primitive in a new context.

Encapsulation cannot be discussed in terms of its algorithms. It requires an extensive knowledge of the local host operating system primitive operations, and a determination of how they can be made to relate automatically to the NSW environment. Thus each TBH approach to encapsulation will probably be different. As far as the other NSW components are concerned, running an encapsulated tool is no different from running a tool which was written to function in the NSW. The behavior of the FE and the WM is identical in the cases of the integrated and non-integrated tool, with the exception that the WM notifies the Foreman in the F$BEGINTOOL message that it will be running a tool which requires encapsulation. We can, however, speak in general terms about certain aspects of encapsulation.

Encapsulation requires some mechanism with which the Foreman can gain control after the tool executes certain operating system functions, but before the operating system proceeds with the local implementation of the operation. The TENEX JSYS trap facility is an example of such a mechanism. It is entirely left to the encapsulation implementation to determine which system calls need trapping and how to integrate these calls with NSW facilities. For the most part, the tool initialization and termination conditions, interactions with the file system, and the communication with the tool user will all require careful attention within the encapsulation component of the Foreman. In some cases, mapping the local system operation into a comparable NSW facility will be straightforward. An example is the terminal interface which drives many tools. The Foreman can simply request the creation of a direct FE connection of type Telnet, and provide this "NSW connection" to the encapsulated tool. In other areas, the Foreman has a wide range of possible implementation strategies. An example of this type is the handling of file delivery into the NSW file system. Since encapsulated tools are not aware of the NSW file system, they cannot guide the Foreman as to the disposition of the files. The Foreman must choose an implementation strategy for delivering new and changed files to the global NSW file space. This can be done as the files become available (i.e., closed in most operating systems), or only at the end of the tool session, or even anywhere in between. Each encapsulation implementation selects the strategy most appropriate for the anticipated needs of the tools for that host.

TENEX NSW encapsulation already exists for selected tools. In general, the simpler a tool is, the more easily it can be encapsulated. By simple, we mean the straightforward use of common operating system facilities. Such facilities are apt to have analogous mechanisms in the NSW, since the NSW caters to many of the same aspects of the tool environment but with wider domain. Depending upon the effort placed into translating system calls, a Foreman will be capable of encapsulating an expanding set of "old tools." However, let us emphasize that encapsulation has limitations. There will always be local host programs which cannot be NSW encapsulated. This is because the NSW system IS different from the local host system, and substituted components can be made to appear similar only to a certain degree. Tools which utilize obscure features, or features peculiar to a particular operating system are sure to be difficult or impossible to encapsulate correctly. Very often this will mean that certain features of a tool are not available when using the tool encapsulated. If this is not satisfactory, or if other problems prevent the tool from being encapsulated (e.g., the local host does not have system facilities for building an encapsulator) then the tool program must be modified to directly call Foreman NSW primitives if it is to function as an NSW tool. Let us also emphasize that for a tool to be most effective in the NSW domain it should be coded using the NSW facilities directly.

We feel that for some TBHs encapsulation can have a high payoff in establishing a large class of programs as NSW tools, and should be actively explored. It is often undesirable to recode existing programs, and it is in this area that encapsulation has its maximum effect. Designing an encapsulator is in many ways similar to designing a tool which directly utilizes the NSW facilities. As such, much of the discussion in the preceding sections of this document will be helpful. As a note of interest, the form of the TENEX encapsulator for the initial test NSW system has influenced the design of the Foreman component, since in a way, the encapsulator was an integrated tool. Some of the concepts embodied in the TENEX encapsulator are discussed as part of Appendix 1, to serve as a model to other encapsulator builders.

IX. Batch Tools

Until the present section, this document has primarily focused on interactive tools. Interactive tools are tools whose users are on-line while the tool is running. There are also batch tools - tools whose users may not be on-line. The primary difference between batch and interactive tools, therefore, is the time at which information about input/output files, parameters, etc. is obtained. A user can supply information to an interactive tool at any time while it is running. A user must supply information to a batch tool before it is run. Thus, an interactive tool needs controlled access to NSW resources while it is running, and an active Foreman supplies the controlled access. Access control for batch tools can be done before the tool is run, so a much less complex Foreman is needed. (Note that nothing in this discussion precludes building a Foreman for batch tools with all of the functions described in this specification. We are merely pointing out that such a complex Foreman is not required.)

We shall sketch the features that are absolutely necessary in a batch Foreman. Since batch tools in NSW will be handled by the IP protocol for the immediate future, we defer details until a later version of this specification.

In the current NSW model, execution of a batch job is handled by the Works Manager Operator (WMO) process. The WMO is given (by the WM) tables which contain skeleton job control language (JCL) and a mapping between dummy parameters in the skeleton JCL and NSW files, real values, etc. The WMO prestages the job on a selected batch host by asking the WM to move (via the File Package) all input files to that host. The skeleton JCL is then edited to insert local file names (obtained as a result of the file movement) and parameter values. The IP server on the batch host is then given the JCL and told to run the job. When the job has run to completion, the IP server informs the WMO, which then moves (again via the FP) the result files into NSW file space. The minimum batch Foreman must support this model. That is, it must have a WMO-invokable F$SUBMITJOB procedure and it must invoke a WMO procedure W$JOBHALT. (In addition, it must support status probes.)

A slightly more complex model requires that the batch Foreman receive the tables now given to the WMO. The batch Foreman would then be responsible for moving input files (either prestaging or during execution) to the batch host, editing the JCL, running the job, moving result files to NSW file space, and informing the WM that the job was complete. We expect that many batch hosts will prefer to control job execution more completely in this later fashion.

Finally, some hosts may choose to implement a complete Foreman. F$SUBMITJOB would then be F$BEGINTOOL and file motion would be handled dynamically. This last case is the least explored of the possibilities although we expect that batch jobs on interactive hosts (TENEX, MULTICS) will be handled by this mechanism.

The WM and WMO will support these several different kinds of batch Foreman so that batch tools may be run on hosts as diverse as B4700, 360/91, and TENEX.

Appendix 1. Functional Summary

This appendix summarizes the externally invocable functions which must be implemented by each Foreman, and proposes parameter value conventions for the functions. Transmissions currently follow the SRI conventions except where noted. That is, transmissions are modified PCPB8 data structures of the form:

LIST (type, length, tid, parameter, args).

This appendix will be modified from time to time, as needed.

Functional Description and Transmission Formats

A. Functions implemented within each Foreman

(Note: Because of a phased implementation plan, and because all functions may not be applicable to all host systems, implementation may consist simply of replying with a rejection message. In that sense, all functions must be implemented (recognized) by all Foreman from the outset. The error code reply value of 177777 (16 bit value) will be taken to mean unimplemented function.)

All functions are invoked with a reply requested (i.e., using TID) except where explicitly stated. Recall that the F$ prefix is used as an expository aid in indicating a function implemented in a Foreman. Where the string "n-" prefixes an element, it should be read as the element repeated n times.

```
A.1  F$BEGINTOOL (program-name, tool-type, entvec, FE-addr,
         cr-addr, filename-list)->result, qstart,
           workspace-descriptor, tool-addr

Program-name:
     charstr:  local host syntax completely specifying the program to
          be run as NSW tool

tool-type:
     index =1 -> encapsulated tool
           =2 -> tool uses NSW calls, does not use MSG
           =3 -> tool uses NSW calls & uses MSG

entvec:
    index =0->do not start tool (illegal except in F$BEGINTOOL)
          =1->continue from point stopped (illegal in F$BEGINTOOL)
          =2->cold start entry point
          =3->warm start entry point
          =4->termination routine entry point
          =7->tool specific entry point
            .
            .
            .
          (Please note that the index value assignments for entvec
           are changed from those indicated in the text of the
           Foreman document.)

FE addr:
   procaddr (new data type corresponding to MSG process address)

cr-addr:
   procaddr
```

```
filename-list:
   list(n-filenames)
         filename:
             charstr:  full local host syntax
result:
    empty -> success
    index -> error code          (error codes to be defined)

qstart:
    boolean = true -> program started
            = false -> program not started

workspace-descriptor:
    list (name, access-info)
               name: charstr
               access-info:  charstr -> info used by File Package
                                    to access workspace via name
                             empty -> access info not needed by File
                                    Package

tool-addr:
     procaddr


A.2  F$STARTTOOL (entvec) -> result

      entvec:  index (see above)
      result:  index (see above)

A.3  F$STOPTOOL (entvec) -> result

      entvec:  index (see above)
      result:  index (see above)

A.4  F$ENDTOOL (reason, termtype, qmaintain)-> result,
                                  accounting-list, qmaintained
     reason:
     index  =1-> user request
            =2-> WM decision
            =3-> user disconnected

     termtype
     index  =1-> no LND processing necessary
            =2-> step thru LND directly with user
            =3-> automatically deliver latest copy of changed files

     qmaintain:
         boolean = true -> maintain tool image if possible
                 = false -> don't maintain tool image (default)

     accounting-list:
         list (cost, n-list(type,amount))
```

```
                 cost: an integer reflecting the cost in cents of
                           running the tool.

                 type: an index indicating the type of resource
                         accounted for
                   =1 -> CPU milliseconds
                   =2 -> connect minutes
                   =3 -> I/O operations
                   =4 -> primitive calls
                   =5 -> core usage
                     .
                     .
                     .  to be defined as needed
                     (note:  each TBH will select the types of
                     resource measures it will provide)
               amount:  integer
                        reflects the session utilization for the
                        corresponding resource type (either in
                        resource units or in cents).

          qmaintained:
             boolean = true -> tool image has been maintained
                     = false -> tool image has not been maintained
                                                     (default)
```

B. Alarms to be recognized by each Foreman

```
B.1  alarm code = 1 -> forthcoming tool termination request
                 response:  immediately begin processing input
                   messages looking for F$ENDTOOL request;
                   all other messages are discarded

B.2  alarm code = 10 -> status probe (name = STATUS)
                 response:  return list (4-statevariables) to
                            invoking process
                   statevariable1: NSW external state
                       index = 0 -> running
                             = 1 -> stopped, never started
                             = 2 -> stopped
                             = 3 -> running at termination code
                             = 4 -> terminated (tool did HALTME)
                   statevariable2: NSW internal state
                       index = 0 -> running
                             = N -> awaiting  completion of NSW
                                    primitive # N
                                    (note:  the primitive name used
                                    by tools need not be uniform
                                    across all TBHs.  However, for
                                    status reports, we standardize
```

```
                                all tool functions by
                                equating each one with a code
                                indicating particular function
                                classes.  It is this code
                                (non-zero) which indicates the
                                type of NSW function the tool
                                is executing).
                 statevariable3:  current local operating system
                                  state
                     index = 0 -> running
                           = 1 -> I/O wait
                           = 2 -> dismissed
                                .
                                .
                                .
                 statevariable4: current program counter
                        integer

     NSW primitive functional classes:
         1-> global file space manipulation
         2-> local file space manipulation
         3-> MSG communication
         4-> tool invocation

B.3  alarm code = 11 -> accounting probe (name = ACCOUNT)
                response:  return accounting-list (defined earlier)
                           to invoking process

                other alarms will be defined as needed
```

A note on responses to alarm codes -

There is currently no convenient way to signify a response to an alarm. Accordingly we are proposing the following addition to the transmission conventions as outlined by Postel.

```
        the standard message transmission format is:
                LIST (type, length, tid, parameter, args)

    for a response to an alarm we specify that

            type = 3 (definition of a new type)
            length   (same as before)
            tid = alarm code (the id field contains the 16 bit
                             alarm code for type 3 messages)
            parameter  (same as for type 2 = acknowledgement)
            args       (same as for type 2)
```

In accordance with this format, a defined alarm code which has no Foreman implementation should return an error reply value of 177777 (unimplemented function).

Receipt of an undefined alarm code can simply be ignored.

C. TENEX Encapsulation

This section, which sketches selected aspects of the TENEX NSW encapsulator, is included as a model for potential encapsulator builders. Encapsulation provides the implementer with large margins of flexibility, and each such implementer must decide upon the nature of an encapsulator best suited for the existing local host programs.

An NSW encapsulated TENEX tool is automatically set up with a network virtual terminal (NVT) to the FE process as its primary input and output device. The structure of the TENEX operating system has allowed the encapsulator to be programmed as an ordinary user process. With respect to the tool it is running, the encapsulator can gain control when the tool executes selected system calls, and in addition can read from and write to the same NVT which was given to the tool.

When a TENEX encapsulated tool requests access to a file (using GTJFN operation), the call is such that in general the program either provides a filename string or indicates a device or file which can be read to obtain the filename. To simplify matters, in this discussion we will consider only the cases where the program has specified that the filename is to be obtained from the NVT (i.e., from the user) or is provided as a text string parameter of the system call. If the call indicates the user as the source of the file name, the encapsulator reads the file name using the NVT. Since the file name has been supplied by the NSW user, we can be certain that it is an NSW syntax Filename referring to an NSW file. As such, we can simply check the LND for a copy of the file, and failing to find one, we can try to obtain a copy from the global workspace. In either of these cases, the tool is provided a direct handle for the TENEX file representing the NSW file copy. In the case of a parameter supplied filename string, the complexity increases. The filename may refer to an NSW file (in NSW syntax) whose name was previously obtained from the user. Alternatively, the filename may refer to a TENEX file (in TENEX syntax) whose use is outside the NSW (e.g., a documentation file). The WM provides a list of such locally accessible files to the encapsulator at tool startup time. Using this list, and based on its knowledge of both the NSW and TENEX file name syntax, the encapsulator determines if the file is a legally referenceable TENEX file or an NSW file. Access to a TENEX file can be granted outright. Once we determine a file name to be an NSW name then we search the LND and consult with the WM as necessary to provide file access.

For those cases where the program requests a "new" file (as indicated by the TENEX system call parameters) it must de facto be an NSW file, since encapsulated tools are not aware of the two different file systems. In this case, an LND entry is created and the encapsulated tool is given a handle on a TENEX file representing the NSW file. TENEX also has the facility to create temporary files, i.e., files which disappear on logout. We have taken the position that tools can utilize temporary files unimpaired, since by their very nature they would not be candidates for being maintained in the central catalog.

Since the encapsulated tools are not reprogrammed for the NSW environment, they do not indicate which files need to be delivered to the WM and when. Thus when an encapsulated tool indicates (in TENEX terms) that it has finished accessing an NSW file, the encapsulator determines whether or not the file has been modified. If it has, then the file is marked in the LND as possibly requiring delivery to the global space at a subsequent time. We have taken the approach that file references are always interpreted locally where possible. That is, in general, file operations will only create new local workspace copies of NSW files and access these copies when they exist. No files are delivered to the global space until the tool terminates, and then only selected files (e.g., latest versions of changed original copies). Version numbering and defaulting are supported as in TENEX, with some assist from the LND. The TENEX encapsulator supports limited TENEX style command editing while accumulating a file name from the user.

Appendix 2. Foreman Induced MSG Additions

The following pages reproduce an appendix to the original MSG design document. The document was originally introduced under the name NSW Note #11. Because of the relevancy to the subject of this document, and because it was not included in the originally distributed MSG report, we are including it as part of the Foreman specification document.

BBN NSW Working Note #11
January 27, 1976

The Impact of the Foreman Concept on MSG

The needs of the Foreman component of the NSW have motivated some proposed additions to the MSG facility. In essence, a Foreman is a local-to-the-tool component of the NSW. The Foreman provides an interface to the tool for the facilities provided by the Works Manager, and in addition helps to provide the NSW environment in which the tool is run. This note discusses one aspect of that environment, the communication facilities made available to the tool.

To a first approximation, the message oriented communication modes provided by MSG to the components that create the NSW environment are also appropriate for tools to communicate with other tools and with the user through a front end process. However tools, especially those in the debugging stage, cannot be allowed to function directly in the uncontrolled MSG environment.

The interprocess communication (IPC) needs of a tool, along with the IPC needs of the Foreman component imply the existence of two logical communication streams. One set of messages is destined for the Foreman, while the other stream is destined for the tool itself. If the IPC needs of a tool can be satisfied using direct connections only, then message traffic can be dedicated to the Foreman implementation. If, on the other hand, the tool must be provided with a message oriented IPC facility which is supported by or derived from the MSG message passing capability, then a multiplexing problem exists. In the following we assume that it is indeed desirable to provide tools with a message oriented communication facility for many of the same reasons that such a facility was desirable for building the NSW itself. We also assume, for obvious reasons, that such a facility will indeed make use of similar MSG functions. Therefore, we must address any problems this situation causes.

For the tool/Foreman complex in the current MSG context there seem only two possibilities: either the Foreman and the tool occupy a single MSG process or they do not. [Note that in any event, the Foreman must maintain a special relationship to the tool. That is, it is the Foreman that provides much of the NSW virtual environment for the tool.] In the case where both the Foreman and the tool occupy the same MSG process, the Foreman is required to receive all incoming messages in order to filter out the ones intended for the Foreman. This filtering would have to be based on NSW addressing conventions transmitted as part of the message data. Messages intended for the tool would be passed to the tool by the Foreman using local operating system facilities outside the scope of MSG. The major advantage of this approach is that it is very convenient to apply the needed access controls on the tool's use of the message facility. The Foreman implements for the tool a new abstract IPC facility which is built upon the Foreman's use of MSG. The "new" IPC facility can be customized for tool use if this is desirable. The major disadvantage for the NSW stems from the fact that the IPC facility we want to provide to the tools is indeed very similar to that offered by MSG. We would like a somewhat restricted version of MSG. Yet to achieve this, we must incur an extra transfer of control between the MSG facility and the Foreman for each incoming and outgoing message of the tool. In addition, this may involve additional handling/copying of the messages and duplicating some of the functions already performed by MSG (e.g. setting up and queuing alarms, handling multiple operations). Furthermore, the Foreman's use of the MSG facility may conflict (interfere) with that of the tool (e.g. MSG queues only a single alarm; also, message sequencing is on an entire MSG process basis). Such conflicts may force an otherwise unnecessary change in the nature of the IPC facility available to tools.

An alternative NSW design is one in which the Foreman and its tool are separate MSG processes. Their distinct MSG addresses would mean that MSG itself could mediate the separation of Foreman messages from those of the tool. The main disadvantage of this approach is that MSG, as currently defined, does not provide any way for the Foreman to limit the tool's use of the message passing facility. The type of control needed for the NSW application is fairly well understood. That is, we would like to be able to limit the conversational partners with which the tool can communicate, and have the Works Manager/Foreman combination responsible for changes in the set of legal conversants. In addition, it may be necessary to forbid a tool from executing certain MSG primitives which are not directly related to the sending and receiving of messages.

In the following sections we outline several additions to MSG, which, if implemented, would allow an NSW tool to directly utilize the MSG IPC facility while allowing the Foreman to specify limitations on how the facility can be used. The MSG additions are in two basic areas. First, we would like to establish the ability of an MSG process to "introduce" to MSG a new process which MSG will then support. Such a new process has presumably been created using whatever facilities are provided by the local host operating system. The result of a process introduction is that the new process is given an MSG process name and can issue MSG primitives. Second, we would provide additional MSG primitives which allow an introducing MSG process to selectively manipulate an access control matrix which would be associated with every introduced process. Such an access control matrix would indicate for each process the allowable objects of each MSG primitive. [In general, the object of an MSG primitive is an MSG process name.]

We view these additions to MSG as the cleanest, most effective way to bring tools into the NSW environment while providing them with a flexible message passing communication facility.

Introducing New MSG Processes

Many modern day operating systems provide mechanisms for dynamic process creation. There has been no limitation placed on the nature of an MSG process, so that multiple processes (in the local host operating system sense) can serve as a single MSG process. In this way MSG programs can continue to utilize any dynamic process creation primitives available on the local host operating system. However, as currently constituted, these MSG processes must share a single MSG address and hence a single MSG message stream. In some cases this is exactly what is desired (e.g. a structure which has one process (in the local system sense) sending all messages, while another does the receiving). However, in other instances (e.g. tool/Foreman) the concurrent, asynchronous operation of multiple processes would be impeded by the single message stream, and force each such MSG process to implement its own local dispatching.

As suggested above, one way to allow MSG itself to mediate the message stream between concurrent but cooperating processes is to assign each a separate MSG address. It may at first seem attractive to add to MSG the notion of a general purpose "create new process". Such a general facility is not necessary for building the NSW. To be sure, such a general inter-host process creation and manipulation mechanism is a goal we have in creating the support environment for the tools themselves, but it need not be implemented by MSG alone. The current discussion is concerned with being able to more flexibly use within the MSG context whatever local host operating system process creation facilities are available. With that goal, there are a number of reasons for refraining from defining a standard MSG "create process" primitive. One is that in many cases the creating process needs to maintain a special relationship between itself and the created process to maintain a particular type of cooperation. This may take the form of being able to directly manipulate certain aspects of the created process, or perhaps results from sharing parts of an address space. In any event, it would be difficult to be able to represent all of the potential relationships from all of the constituent systems, and even more difficult to implement some of them. A second argument against a standard MSG "create process" primitive is that it would have to be accompanied by a suitable way of describing the process that was to be created. Typically this is handled in the context of a file system, but there does not exist a unified MSG file store. [Although a unified multi-host file system is part of the NSW design, it is not realized at the MSG level.]

An alternative to a unified MSG "create process" primitive is the approach which acknowledges the local nature of the creation and specification of new processes, but allows the creating process to "introduce" to MSG the created process. Any special relationship between the processes, as well as the means of specifying how to create the process is handled on a strictly local host basis. It is presumably complete before the introduction is made.

After introducing a new process to MSG, we will have established two separate MSG addresses. The Foreman and tool can have separate MSG message streams, with MSG mediating between them. However, process introduction by itself does not solve all of the problems raised by the NSW Foreman application.

Limiting the Use of MSG Facilities

Up until this point, the MSG documentation has been careful to avoid specifying any hierarchy of control among the MSG processes. The MSG processes are largely independent of one another from a control standpoint. Communication between processes via MSG messages is unrestricted, with each process determining for itself the validity of any message it receives. The concept of "introducing" another process into MSG establishes a natural place for integrating a form of control by selected MSG processes over other MSG processes. The introduction of any such control is again motivated by the Foreman application. However, we attempt to define the use of the control facilities in a way which does not preclude its application in other contexts.

As a result of process introduction, the introducing process is established as the superior of the introduced process. The superior/inferior relationship represents a tight coupling among MSG processes, as contrasted to the loose coupling provided through message communication. Termination of a superior MSG process also causes the termination of its inferiors. Superior processes are permitted to regulate their inferior's use of the MSG facilities. This is accomplished through the addition of a control mechanism associated with the execution of MSG primitives. In general terms, the control consists of associating an access control matrix with each new MSG process, and allowing superior processes to manipulate (through MSG primitives) the contents of the matrix for their inferiors. Entries in the matrix represent permission to execute a particular MSG primitive on a particular object. MSG will reject a primitive call from one of its processes if the access control matrix does not indicate that this (call, object) pair is allowable. An object is usually an MSG process name. However, some MSG primitives (e.g. Stopme) do not take objects as arguments. In such cases, a single entry regulates the ability to execute such a primitive.

It may be helpful to view an MSG process which has no MSG superior (i.e. has not been "introduced" by another MSG process) as having an unrestricted access control matrix. An MSG process can only supply its inferiors with rights that it currently has. Removal of a right from an immediate inferior causes removal of that right from any MSG process further down the hierarchy. Applying access control to the sending of messages (data) has the beneficial side effect of reducing bandwidth consumption by unauthorized messages. It also increases the confidence in the validity of messages which are received.

MSG Primitives

The following is a set of primitives which, if added to MSG, would allow the Foreman to function in the previously discussed mode. (This is just a rough sketch of the primitives, along with some possible implementation details).

1. Introduce New Process (pointer to process descriptor, pointer to initial access control information, location of returned MSG name, disposition)

This is the primitive which is used to introduce a new process into MSG. In response to this primitive, MSG establishes an MSG address for the introduced process. The generic component of the generated name is always null, implying MSG has no knowledge of the function performed by the process.
It also establishes the issuing MSG process as the superior of the process, and initializes the access control matrix based on the data passed in the parameter list. The new MSG name is returned to the calling process.

process descriptor: local host operating system dependent parameter for conveying to MSG the identity of the new process. The exact nature of this parameter is dependent on the entity which is discernable as a process on the local operating system.

access control information: list of pairs, where each pair is of type (primitive, list of objects). Primitive denotes a particular MSG primitive, and list of objects is a list of MSG process names. Special designations exist for the classes of objects "all" and "none". Individual process names include all defined MSG process name fields, with the addition that each field may optionally have the "all" designator.

2. Renounce Process (MSG process address, disposition)

This primitive is the inverse of process introduction. MSG checks to see if the issuing process is the superior of the object process, and if so MSG causes the removal of the object process (including removing any knowledge of its existence from MSG tables). This also forces immediate rejection of all outstanding MSG operations on behalf of the object process. Any MSG processes introduced by the object process are similarly renounced. Note that this does not have to necessarily result in the destruction (in the local operating system sense) of whatever constituted the MSG process. It only makes MSG itself unaware of its existence.

3. Update Control Table (MSG process name, add/delete indicator, pointer to access control information, disposition)

This primitive is used to manipulate the access control information associated with an inferior MSG process. MSG checks to see if issuing MSG process is the superior of the object process, and if so updates the access control matrix of the object process according to the supplied parameters. In the case of additions, the (primitive, object) pair specified must be currently accessible to the issuing process in order for this update to succeed. A deletion causes the same pair to be removed from any inferiors of the object MSG process.

access control information: same as defined in the introduce primitive.

add/delete: boolean which distinguishes adding entries from removing them.

REPORT DOCUMENTATION PAGE — READ INSTRUCTIONS BEFORE COMPLETING FORM

14

1. REPORT NUMBER: BBN ~~Report No.~~ -3266
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER: 9

6

4. TITLE (and Subtitle): The Foreman: Providing the Program Execution Environment for the National Software Works.
5. TYPE OF REPORT & PERIOD COVERED: Scientific rept.,
6. PERFORMING ORG. REPORT NUMBER:

10

7. AUTHOR(s): Richard E. Schantz, Robert E. Millstein
8. CONTRACT OR GRANT NUMBER(s): 15 N00014-75-C-0773, F30602-76-C-0094
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek and Newman Inc., 50 Moulton Street, Cambridge, Massachusetts 02138
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 16 ARPA Order-2901
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, Information Processing Techniques, 1400 Wilson Blvd., Arlington, VA
12. REPORT DATE: 3/ March 1976 11
13. NUMBER OF PAGES: 57
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Code ONR-430D, 800 N. Quincy Street, Arlington, Virginia 22217
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public. 12 63p.
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
    (a) This research supported by DARPA under ARPA Order No. 2901
    (b) This report also published by Massachusetts Computer Associates, Document No. CADD-7604-0111.
19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

National Software Works	Tool Bearing Host
Computer Networks	Network Protocols
Distributed Systems	Network File System
Resource Sharing	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the function and operation of the National Software Works (NSW) system component known as the tool Foreman. A tool is a software development computer program made available to users through the multi-computer NSW system. The Foreman provides tools with their NSW execution environment, and ensures the smooth operation of tools within the NSW. Various aspects of the tool environment and the Foreman interface with other NSW system processes are described.